

Figure 1
(Prior Art)

117
MAGIED M. MICHAEL et al
YOR 920030768 USA (NSJ) 8128-632

200

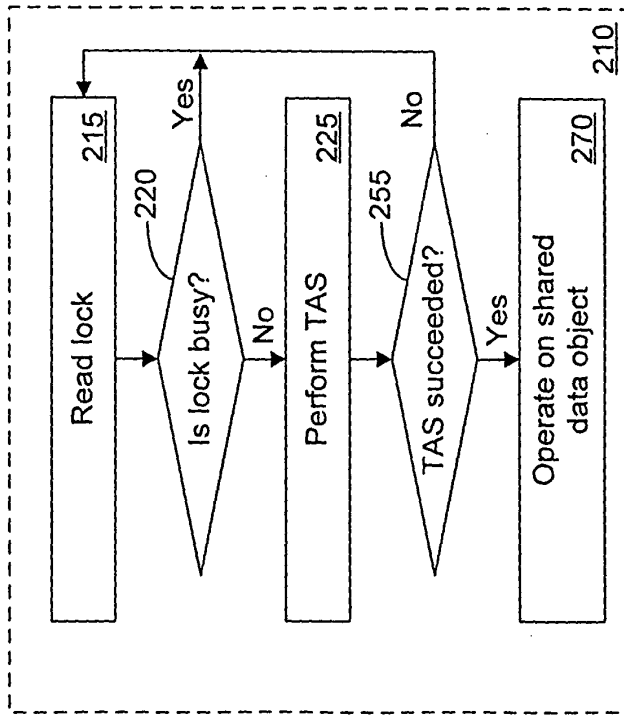


Figure 2A
(Prior Art)

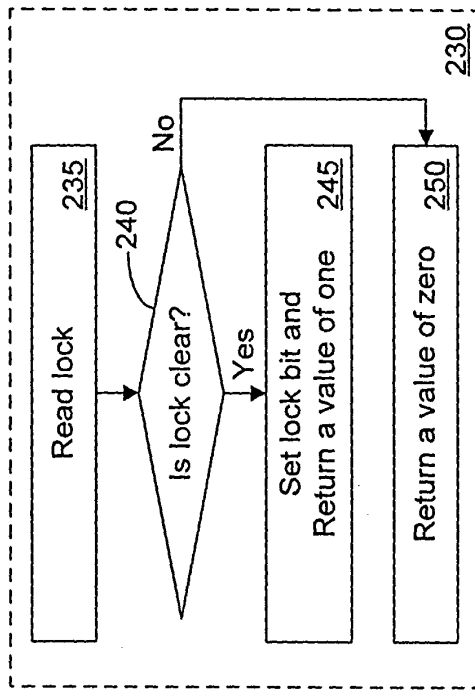


Figure 2B
(Prior Art)

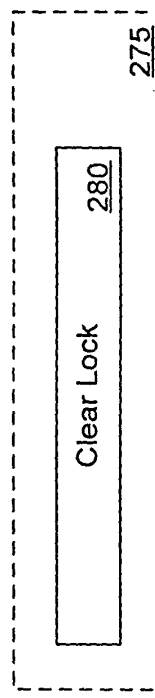


Figure 2C
(Prior Art)

217
4DR92DD2D268US2 (8728-632)

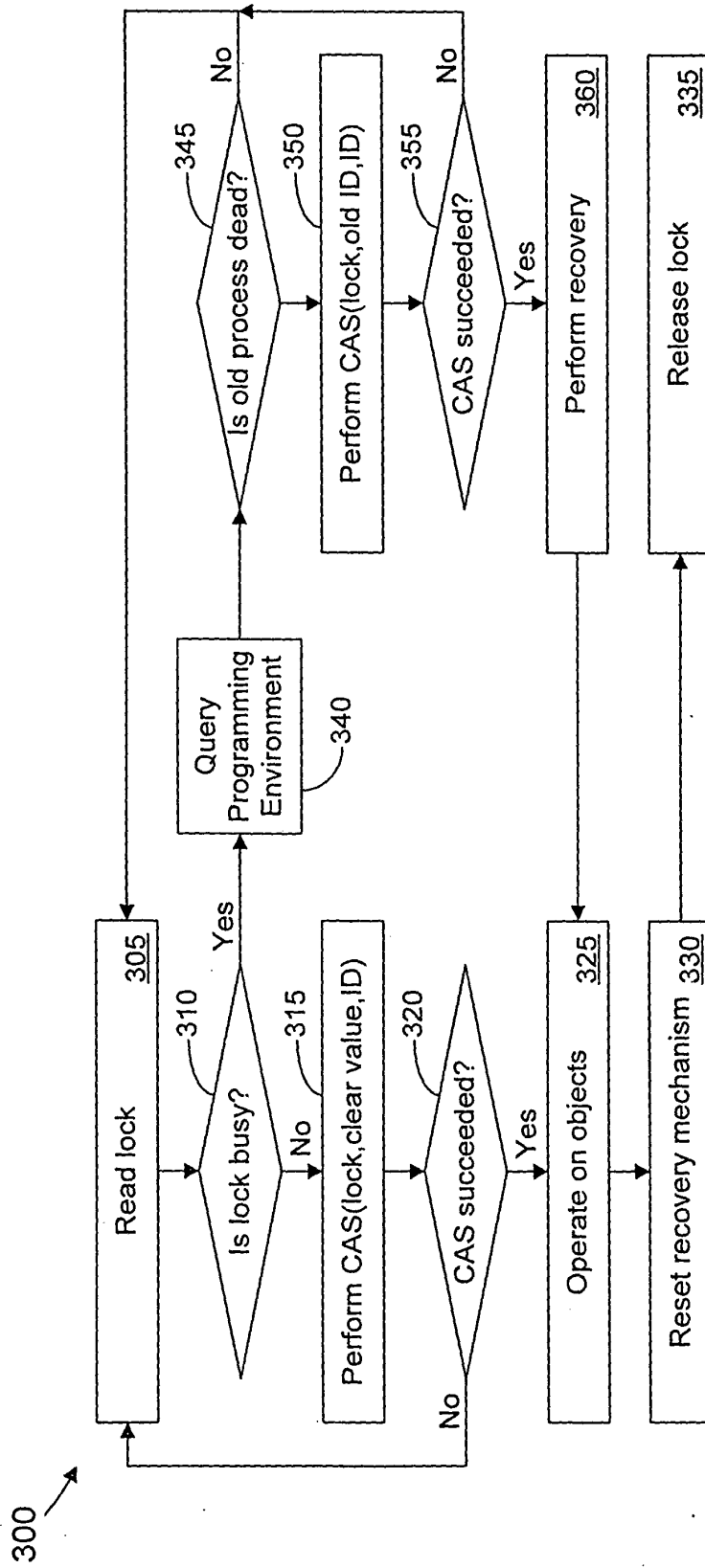


Figure 3

4/7
YDR920030268US1 (8728.632)

```

// Lock variables
LockHolder : ProcessIdType;
LastChecked : TimeType;

AcquireLock (p : ProcessIdType)
while true {
A1:   if (holder ← LockHolder) = NULL {
A2:     if CAS(&LockHolder, NULL, p)
A3:       {LastChecked ← Time(); return; }
    } else {
A4:       last ← LastChecked;
A5:       time ← Time();
A6:       if time - last < THRESHOLD continue;
A7:       if ¬CAS(&LastChecked, last, time) continue;
A8:       if QueryOS(holder) = ALIVE continue;
A9:       if CAS(&LockHolder, holder, p)
A10:        {UserRecovery (...); return; }
    }
}

ReleaseLock ()
R1: ResetRecovery (...);
R2: LockHolder ← NULL;

```

Figure 4

```

structure NodeType
    Pid : ProcessIdType;
    Status : (WAITING, HASLOCK, FAILED), TagType;
    Next : (*NodeType, TagType);
    LastChecked : TimeType;
    Head, Tail : (*NodeType, TagType) Initially NULL;
    LockHolder : *NodeType Initially NULL;

    AcquireLock (node : *NodeType)
    A1: Enqueue(node);
    A2: old_head ← Head.Data; ptr ← old_head;
    while true {
    A3: if WaitForSignal () = HASLOCK {
    A4: node.LastChecked ← Time0;
    A5: LockHolder ← node; return;
    }
    // Timeout is detected.
    A6: holder ← LockHolder;
    A7: if (head ← Head.Data) ← old_head {
    A8: old_head ← head; ptr ← head;
    A9: } else ptr ≠ node {
    A10: ptr ← ptr.Next.Data;
    } else {
    A11: old_head ← head; ptr ← head;
    A12: if holder ≠ NULL ∧ holder ≠ head {
    A13: if ~ProcessFailed(holder, head) continue;
    A14: if holder ≠ LockHolder continue;
    }
    A15: if UsurpLock (node, head) return;
    }
    }

    ReleaseLock (node : *NodeType)
    R1: ResetRecovery (.);
    R2: next ← Dequeue(node);
    R3: if next ≠ NULL <status, p> ← next.Status;
    R4: CAS(&LockHolder, node, NULL);
    R5: if next ≠ NULL
        CAS(&next.Status, <status, p>, <HASLOCK, t + 1>);

    ProcessFailed(ptr, head : *NodeType) : boolean
    while true {
    Q1: if ptr.Status = <FAILED, 0> return true;
    Q2: if head ≠ Head.Data return false;
    Q3: last ← ptr.LastChecked;
    Q4: time ← Time0;
    if time - last ≤ THRESHOLD continue;
    Q5: if DCAS(ptr.LastChecked, last, time) continue;
    Q6: if QueryOS(ptr.Pid) = ALIVE return false;
    Q7: node.Status ← <FAILED, 0>; return true;
    }

    WaitForSignal (node : *NodeType) : (HASLOCK,
    TIMEOUT)
    W1: last ← Time0;
    while true {
    W2: status ← node.Status.Data;
    W3: if status = HASLOCK return HASLOCK;
    W4: time ← Time0;
    W5: if time - last > THRESHOLD return TIMEOUT;
    }

    UsurpLock (node, head : *NodeType) : boolean
    U1: ptr ← head;
    while ptr ≠ node {
    U2: if ~ProcessFailed(ptr, head) return false;
    U3: ptr ← ptr.Next.Data;
    U4: if head ≠ Head.Data return false;
    }
    U5: Head ← <node, Head.Tag + 1>;
    U6: if node ≠ head UserRecovery (.);
    U7: node.Status ← <HASLOCK, node.Status.Tag + 1>;
    U8: LockHolder ← node; return true;

    Enqueue(node : *NodeType)
    E1: node.Next ← <NULL, node.Next.Tag + 1>;
    while true {
    E2: <tail, tp> ← Tail;
    E3: <head, tp> ← Head;
    E4: if tail ≠ NULL ∧ head ≠ NULL {
        <next, tp> ← tail.Next;
        if next = NULL {
            if Tail ≠ <tail, tp> continue;
            node.Status ← <WAITING, node.Status.Tag + 1>;
            if ~CAS(&tail.Next, <NULL, tp>, <node, tp + 1>)
                continue;
            CAS(&Tail, <tail, tp>, <node, tp + 1>);
            return;
        } else if ~CAS(&Tail, <tail, tp>, <NULL, tp + 1>)
            continue;
        CAS(&Head, <head, tp>, <NULL, tp + 1>);
        CAS(&Tail, <tail, tp>, <next, tp + 1>);
    } else if ~CAS(&Tail, <tail, tp>, <NULL, tp + 1>)
        continue;
    } else
        CAS(&Tail, <tail, tp>, <next, tp + 1>);
    } else if tail = NULL ∧ head = NULL {
        node.Status ← <HASLOCK, node.Status.Tag + 1>;
        if CAS(&Tail, <NULL, tp>, <node, tp + 1>) return;
        if self Tail = <tail, tp>
            CAS(&Head, <head, tp>, <tail, tp + 1>);
    }
}

Dequeue(node : *NodeType) :
*NodeType
D1: <next, tp> ← node.Next;
D2: <tail, tp> ← Tail;
if next = tail {
    if next = NULL {
        if CAS(&node.Next,
        <NULL, tp>,
        <DEQUEUED, tp + 1>) {
            CAS(&Tail, <node, tp>,
            <NULL, tp + 1>);
            tp ← Head.Tag;
            CAS(&Head, <node, tp>,
            <NULL, tp + 1>);
            return NULL;
        }
        <next, tp> ← node.Next;
    }
    CAS(&Tail, <node, tp>, <next,
    tp + 1>);
}
D9: Head ← <next, Head.Tag + 1>;
return next;
}

```

Figure 5

517
Y09200302268451 (8728-632)

617
YD0920030268451 (8728-632)

```

structure NodeType
  Pid : ProcessIdType;
  Status : {WAITING, HASLOCK, FAILED, REMOVED,
    TO_BE_REMOVED, TagType};
  Next : (*NodeType, TagType);
  LastChecked : TimeType;
  Signal : *NodeType;
  Ack : boolean;
  Head, Tail : (*NodeType, TagType) initially NULL;
  LockHolder : *NodeType initially NULL;

  PS-AcquireLock(node : *NodeType)
  a1: repeat until TryLock (node) = HASLOCK;

  PS-ReleaseLock(node : *NodeType)
  r1: ReexitRecovery (...);
  r2: next ← Dequeue(node);
  while next = NULL {
  r3: <status, p> ← next.Status;
  r4: if CheckPreemption(node, next) = ACTIVE {
  r5: CAS(&LockHolder, node, NULL);
  r6: CAS(&next.Status, <status, p>, <HASLOCK, p + 1>);
  r7: return;
  }
  r8: next.Status ← <TO_BE_REMOVED, p + 1>;
  r9: p ← Dequeue(next);
  r10: CAS(&next.Status, <TO_BE_REMOVED, p + 1>, <REMOVED, p + 2>);
  next ← p;
  }
  r11: CAS(&LockHolder, node, NULL);

  CheckPreemption(node, next : *NodeType) : {ACTIVE, PREEMPTED}
  c1: node.Ack ← false;
  c2: next.Signal ← node;
  c3: for i ← 0 to PREEMPTION_THRESHOLD
  c4: if node.Ack return ACTIVE;
  c5: return PREEMPTED;

  TryLock (node : *NodeType) : {HASLOCK, REMOVED}
  t1: Enqueue(node);
  t2: old_head ← Head.Data; p ← old_head;
  while true {

```

```

t3: if (status ← PS-WaitForSignal()) = HASLOCK {
t4: node.LastChecked ← Time0;
t5: LockHolder ← node; return HASLOCK;
}
t6: else if status = REMOVED return REMOVED;
t7: holder ← LockHolder;
t8: head ← Head.Data;
t9: <status, p> ← node.Status;
  if status = TO_BE_REMOVED ∧ node ≠ head
t10: {node.Status ← <REMOVED, p + 1>; return REMOVED; }
t11: if head ≠ old_head {
t12: old_head ← head; p ← head;
t13: } else if p ≠ node {
t14: p ← p.Next.Data;
  } else {
t15: old_head ← head; p ← head;
t16: if holder ≠ NULL ∧ holder ≠ head {
t17: if ~ProcessFailed(holder, head) continue;
t18: if holder ≠ LockHolder continue;
  }
t19: if UsurpLock (node, head) return HASLOCK;
}
}

PS-WaitForSignal(node : *NodeType) : {HASLOCK, REMOVED, TIMEOUT}
w1: last ← Time0;
w2: node.Signal ← NULL;
  while true {
w3: if (p ← node.Signal) = NULL {
w4: p.Ack ← true;
w5: node.Signal ← NULL;
  }
w6: status ← node.Status.Data;
w7: if status ∈ {HASLOCK, REMOVED} return status;
w8: time ← Time0;
w9: if time - last > THRESHOLD return TIMEOUT;
  }

```

Figure 6

// MAX is the maximum number of updates in a CS
structure *LogType*

count : 0..MAX; // initially 0
addr[MAX] : pointer;
prev[MAX] : *ValueType*;

WriteAndLog(addr : pointer, v : ValueType)

log.prev[*log.count*] \leftarrow **addr*;
log.addr[*log.count*] \leftarrow *addr*;
log.count \leftarrow *log.count* + 1;
**addr* \leftarrow *v*;

UserRecovery ()

for *i* \leftarrow *log.count* - 1 **downto** 0
 **log.addr*[*i*] \leftarrow *log.prev* [*i*];
log.count \leftarrow 0;

ResetRecovery ()

log.count \leftarrow 0;

Figure 7